

2

DTIC FILE COPY

NPS-53-88-007

NAVAL POSTGRADUATE SCHOOL

Monterey, California

AD-A198 354



DTIC
ELECTE
AUG 3 1 1988
S H D

A TUTORIAL ON APL2

by

Toke Jayachandran

Technical Report for Period
September 1987 - July 1988

Approved for public release; distribution unlimited

Prepared for:
Naval Postgraduate School
Monterey, CA 93943

88 8 21 031

NAVAL POSTGRADUATE SCHOOL
Department of Mathematics

R. C. AUSTIN
Rear Admiral, U.S. Navy

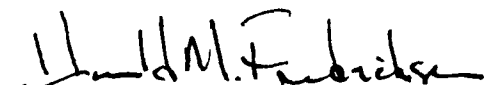
HARRISON SHULL
Provost

Reproduction of all or part of this report is authorized.



TOKE JAYACHANDRAN
Professor of Mathematics

Reviewed by:



HAROLD M. FREDRICKSEN
Chairman
Department of Mathematics



KNEALE T. MARSHALL
Dean of Information and
Policy Sciences

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b RESTRICTIVE MARKINGS		
2a SECURITY CLASSIFICATION AUTHORITY			3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited		
2b DECLASSIFICATION/DOWNGRADING SCHEDULE					
4 PERFORMING ORGANIZATION REPORT NUMBER(S) NPS-53-88-007			5 MONITORING ORGANIZATION REPORT NUMBER(S) NPS-53-88-007		
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School		6b OFFICE SYMBOL (if applicable) 53		7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School	
6c ADDRESS (City, State, and ZIP Code) Monterey, CA 93943			7b ADDRESS (City, State, and ZIP Code) Monterey, CA 93943		
8a NAME OF FUNDING/SPONSORING ORGANIZATION Naval Postgraduate School		8b OFFICE SYMBOL (if applicable) 53		9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c ADDRESS (City, State, and ZIP Code) Monterey, CA 93943			10 SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO	PROJECT NO	TASK NO.
11 TITLE (Include Security Classification) A Tutorial On APL2					
12 PERSONAL AUTHOR(S) Toke Jayachandran					
13a TYPE OF REPORT Technical Report		13b TIME COVERED FROM 9/1/87 TO 7/31/88		14 DATE OF REPORT (Year, Month, Day) 1 August 1988	
15 PAGE COUNT 37					
16 SUPPLEMENTARY NOTATION					
17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number) programming language, APL, APL2, tutorial		
FIELD	GROUP	SUB-GROUP			
19 ABSTRACT (Continue on reverse if necessary and identify by block number) This report contains a short tutorial on the new features of the APL language processor called APL2, available on the NPS mainframe computer.					
20 DISTRIBUTION/AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/INLIMITED <input type="checkbox"/> SAME AS RPT <input type="checkbox"/> DTIC USERS				21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a NAME OF RESPONSIBLE INDIVIDUAL Toke Jayachandran				22b TELEPHONE (Include Area Code) (408) 646-2600	
				22c OFFICE SYMBOL 53Jy	

A TUTORIAL ON APL2

INTRODUCTION

APL2 is an advanced APL language processor which is essentially a superset of the older VSAPL processor, both available on the NPS mainframe computer. Practically all of the commands and functions in VSAPL perform exactly the same way in APL2 and several new features that enhance the programming and data processing capabilities are included in APL2. Both VSAPL and APL2 will continue to be available to the user. The aim of this report is to provide a short tutorial on the new features in APL2; the reader is assumed to be familiar with the APL language and the VSAPL processor. Two IBM publications (references [1] and [2]) and the recently published book "APL2 At a Glance" (reference [3]) provide a comprehensive discussion of the capabilities of the APL2 processor.

A minimum of 1.5 megabytes of virtual memory is required in order to use APL2. The CMS command: GETSTOR 1500K followed with an ENTER will assign the requisite amount of memory. The APL2 processor may now be invoked with the command:

APL2

The response from the system might be as follows:

```
APL2  1.2.00  (ENGLISH)
PROGRAM PRODUCT NUMBER 5668-899
VERSION 1, RELEASE 2
```

```
CLEAR WS+
```

The + after WS indicates that a system message describing some feature of the processor, may be displayed with the command:)MORE. For specialized applications it is possible to add certain keyword options to the invocation command e.g., APL2 OPTION1 OPTION2 . . . The available options and their definitions are described in [1]. As with VSAPL, the GRAFSTAT package can be used from within APL2; the appropriate invocation command for this application is APL2GS instead of APL2.

Workspaces created under APL2 will have file type APLWSV2 as compared to VSAPLWS for the older APL workspaces. These older VSAPL workspaces cannot be loaded into APL2 with the)LOAD command and the)LIB command will not even list them. However, it is possible to convert them into APL2 workspaces by first "migrating" them into APL2 with the command:)MCOPY WNAME and then saving them with the)SAVE WNAME command. The APL2 workspace name can be the same as the VSAPL workspace; because of the differences in the file types both workspaces will still exist on the A-disk. The APL2

command: JOFF returns the user to the CMS environment; there is no command to directly log off from the APL2 environment.

Complex arithmetic can be carried out in APL2. A complex number of the form $a+ib$ is entered as aJb or in its polar form as $mD\phi$ or $mR\phi$ where m is the magnitude of the complex number and ϕ is its angle measured in degrees or radians. If R is a complex number, $+R$ is its conjugate, $|R$ is its magnitude and $\times R$ represents a complex number with magnitude 1. The APL2 primitive functions $+$, $-$, \times and \div perform the standard complex arithmetic in the usual way.

ARRAYS IN APL2

A very useful capability available in APL2 is the ability to create "mixed arrays" and "nested arrays". A mixed array is one that contains both numeric and character constants in the same array. A nested array can have other arrays as its basic elements; for example, a nested array could be a matrix each of whose elements is a vector or even another matrix. The following examples illustrate the creation of mixed and nested arrays.

EXAMPLES:

$X1 \leftarrow 1 \ 2 \ 3 \ 4$

$X2 \leftarrow (1 \ 2)(3 \ 4)$

$Y1 \leftarrow 'ABCD'$

$Y2 \leftarrow 'A' \ 'B' \ 'C' \ 'D'$

$Y3 \leftarrow 'AB' \ 'CD'$



Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

Note that it requires separate commands to create the arrays X1, X2, Y1, Y2 and Y3. X1 is a 4-element array of numeric constants. X2 is a nested 2-element array whose both elements are themselves 2-element arrays. Y1 and Y2, although defined differently, represent the same array i.e., a 4-array of character scalars and Y3 is a nested 2-array with 2-arrays as its elements.

```
Z←(2 3)ρ(14) 'ABCD' '****' (5 6 7 8J9) 'EFGH' 'ΔΔΔΔ'
W←'ONE' 'TWO' ('BUCKLE'('MY' 'SHOE'))
D←2 2 ρ'ONE' 'TWO' 'BUCKLE' ('MY' 'SHOE')
```

Z is a mixed-nested array, a matrix whose three elements in the first row are the arrays (1 2 3 4), (ABCD), (****) and whose three elements in the second row are (5 6 7 8J9), (EFGH), (ΔΔΔΔ). W is a 3-element nested array in which the first two elements are 3-element arrays and the third element is itself a 2-element nested array. When displayed on the screen these objects will appear as follows:

```
X1↔1 2 3 4      X2↔1 2  3 4      Y1↔Y2↔ABCD      Y3↔AB CD

Z↔1 2 3 4      ABCD  ****
      5 6 7 8J9  EFGH  ΔΔΔΔ

W↔ONE TWO      BUCKLE  MY SHOE
```

These arrays will be used repeatedly to demonstrate various concepts; the reader may find it convenient to create them in a clear workspace and try out some of the new functions and commands.

Notice the number of spaces between the elements of W. There is one space between ONE and TWO and also between MY and SHOE; there are two spaces between BUCKLE and MY and three spaces between TWO and BUCKLE. The spacing is designed to indicate the levels of nesting in the array; however, the only foolproof method for determining the degree of nesting in an array is to use the new APL2 function DISPLAY described below.

Public Library 1 includes a workspace called DISPLAY that contains an APL function, also called DISPLAY; this function will present a pictorial representation of the degree of nesting in a given array, on the screen. To use this function it must first be copied into the active workspace with the command:

```
)COPY 1 DISPLAY DISPLAY or )COPY 1 DISPLAY
```

In APL2 it is possible to create several different arrays with a single command. Also, multiple usage of evaluated input ($A \leftarrow \square \square$) or character input ($B \leftarrow \square \square \square$) is allowed.

$(AA \ BB) \leftarrow 10$ will assign to both AA and BB the scalar 10.

`(AA BB)←(1 2 3)'XYZ'` will assign (1 2 3) to AA and XYZ to BB.

It is not necessary to separate objects enclosed in parentheses with spaces.

`10+(AA BB)←100` will create AA and BB both containing 100 and then display 110, the result of the addition `10+100`.

`(A B C)←⎵ ⎵ ⎵` will display three successive ⎵: for keyboard input. The three inputs will be assigned to C, B, A in that order; same for ⎵ also.

A number of new primitive functions and operators that are especially designed to handle nested arrays are included in APL2. Recall that a primitive function is one that is invoked by entering a single APL character such as ρ or Φ . An APL "operator" takes as its operand an APL function, to produce a new user defined function; for example, the primitive operator `" / "` (reduction) when combined with the primitive function `" + "` results in the summation function `" +/ "`. A new primitive operator called EACH invoked with the APL character `" ⋄ "` (dieresis - Shift 1 on the APL keyboard) is particularly useful for manipulating nested arrays. When this operator is combined with a primitive function as its left operand the result is a new function whose effect is to apply the original primitive function individually to each of the elements of its argument array. Thus, `⋄⍴` when applied to an array will display the shapes of each of the elements in the array

separately.

```

+/(1 2)(3 4)(5 6)↔9 12      +/'(1 2)(3 4)(5 6)↔3 7 11

ρ''X1↔EMPTY   ρ''X2↔2 2      ρ''Z↔4 4 4      ρ''W↔3 3 2
                                4 4 4

```

Notice that the third element of ρ''W is 2 since this element is a nested 2-element array with components BUCKLE and MY SHOE.

DEPTH is a new primitive function invoked with the APL character " = ". On the newer IBM 3179-G2 terminals the character can be generated with a single key stroke; the rightmost key in the keyboard row ASDFGHJKL:"=. On older terminals such as the IBM 3278s it is an "overstruck" character; the system command:)PBS ON described on page 24, must be used to generate the character. DEPTH is a monadic function that displays the degree of nesting in its argument array. The DEPTH of a simple scalar (numeric or character) is 0 and the DEPTH of any non-nested array is 1 regardless of its dimension (a vector or a matrix or any higher dimensional array). For nested arrays, the DEPTH is 1 plus the depth of the item with maximum depth. Thus,

```
=X1↔1,  =X2↔2,  =Z↔2,  =W↔4,  =D↔3,  ='W↔1 1 3
```

```
and      ='W ↔ 0 0 0 0 0 0 1 2
```

The dyadic function PICK (APL character \triangleright , Shift X) will select the element specified in the left argument from an array specified as its right argument. For example,

```
2>X1 $\leftrightarrow$ 2          2>X2 $\leftrightarrow$ 3 4          2 2>X2 $\leftrightarrow$ 4
3>W $\leftrightarrow$ BUCKLE MY SHOE      3 2>W $\leftrightarrow$ MY SHOE      3 2 2 4>W $\leftrightarrow$ E
```

For simple arrays, a single number as the left argument of PICK will suffice to identify the object to be picked. For nested arrays, if the left argument is a single number the entire nested element at the specified location will be picked; if the left argument is an array all but the rightmost number identify the location of a nested element from which an object is to be picked and the rightmost number is the indicator of the location of the specific object to be picked. Thus, 3 2 2 4>W will pick the fourth character 'E' from the 2nd sub-element 'SHOE' of the 2nd sub-element 'MY SHOE', which is in turn the 2nd sub-element of the third element of W, 'BUCKLE MY SHOE'.

Picking objects from matrices and higher dimensional arrays is a little more involved and will be discussed after the introduction of ENCLOSE a new APL2 function.

ENCLOSE is a monadic function (APL character \leftarrow , Shift Z) that artificially converts any array into a single "scalar-like" object whose shape would, of course, be empty. If X is any array, $\leftarrow X$ is a scalar object i.e., $\rho \leftarrow X \leftrightarrow \text{EMPTY}$.

The ENCLOSE function is necessary for picking objects from a matrix or a higher dimensional array. To PICK the entire element in a specified row and specified column of a matrix (this object could be an array) the location indicator must first be converted into a scalar object using ENCLOSE.

(c2 2)>D ↔MY SHOE picks the (2,2) element of D which is a nested array. However, to PICK the 2nd sub-element SHOE the appropriate command is

(2 2)2>D ↔SHOE When picking sub-elements from an element at a given location, the location indicator is specified within parentheses but is not converted into a scalar using ENCLOSE.

Thus, to PICK an element from a matrix or a higher dimensional array, the item to be picked must be preceded by its location (row,column etc.) specified between parentheses and the element indicator follows the location indicator; if the entire item at a given location is to be picked the ENCLOSE function must be applied to the location indicator. PICK can be very useful for replacing or modifying elements in an array without the necessity for redefining the entire object.

For the array X1↔1 2 3 4 X1[3]↔3 and
X1[3]←c'THREE' will replace the number 3 with the
the scalar like object THREE. X1 is
now a mixed-nested array. However,

X1[3]←'THREE' will result in an error since this command calls for the replacement of a scalar object with an array.

((2 2)2>D)←'BELT' will replace SHOE with BELT in array D.
This command calls for the replacement of the 2nd sub-element in the 2nd row and 2nd column of D with BELT.

When using the ENCLOSE function one can also specify an axis of a higher dimensional array along which the function to be applied e.g., <[1]R or <[2]R.

Let A←2 3ρ16 and B←(2 4ρ18) 9 (3 2ρ'ABCDEF'). Then,

A←→1 2 3	B←→1 2 3 4	9 AB	ρB←→3
4 5 6	5 6 7 8	CD	ρ"B←→2 4 3 2
		EF	

<[1]A←→1 4 2 5 3 6 The ENCLOSE function is applied to the columns of A to create a 3-array of scalar-like objects; note that ρ<[1]A=3 and ρ" <[1]A is EMPTY.

<[2]A←→1 2 3 4 5 6 a 2-array with rows of A as elements.

The new monadic function DISCLOSE (APL character \triangleright , shift X on the APL keyboard) has many uses. It can be used as an inverse of ENCLOSE to negate its effect. If X is an array, $\triangleright X \leftrightarrow X$ i.e., the combination of ENCLOSE followed by DISCLOSE will have no effect on an array X. Another important use for DISCLOSE is in the creation of tables. Suppose the 3-element mixed-nested array MENU is defined by

```
MENU←('HAMBURGER' 2.00)('FRIES' 1.00)('COKE' 0.75)
```

Then, the command: \triangleright MENU will convert the array into a table that will be displayed as below.

HAMBURGER	2.00
FRIES	1.00
COKE	0.75

The above example illustrates how the DISCLOSE function can simplify the task of creating tables and charts. DISCLOSE can be applied to an array R of any dimension; the only requirement is that all the elements of R are either scalars or themselves arrays of the same rank but not necessarily of the same shape. DISCLOSE has no effect on simple (non-nested) arrays. For nested arrays its effect is to create a new array which is at least one dimension higher than the original array R. The sizes of the newly created dimensions are directly related to the sizes of the sub-arrays of R. Let the nested 3-array B be defined by

B←(2 4ρ18) 9 (3 2ρ'ABCDEF') . Then,

ρB↔3 ρ"B↔2 4 3 2

ρB is a three dimensional array of shape
(3 3 4) whose 3 components along the
first dimension are

(ρB)[1; ;]↔1 2 3 4	(ρB)[2; ;]↔9 0 0 0	(ρB)[3; ;]↔A B
5 6 7 8	0 0 0 0	C D
0 0 0 0	0 0 0 0	E F

The size of the first dimension, 3, is the number of elements in B, the size of the second dimension is 3, the largest number of rows in the elements of B, and the third dimension, 4, is determined by the maximum number of columns in the elements of B. Note that numerical objects are padded with zeros and character objects are padded with spaces (ρ(ρB)[3; ;]↔3 4) to make them conform to size requirements. The following examples illustrate the use of DISCLOSE with axis specification. Consider the nested-mixed matrix Z defined earlier.

ρZ↔2 3 4	(ρZ)[1; ;]↔1 2 3 4	(ρZ)[2; ;]↔5 6 7 8J9
	A B C D	E F G H
	★ ★ ★ ★	△ △ △ △

The shape of Z, (2 3), determines the first two dimensions

of $\supset Z$ and the size of the newly added dimension, 4, is the shape of the largest sub-array of Z.

```

p>[1]Z←4 2 3  (>[1]Z)[1;;]←1 A*      (>[1]Z)[2;;]←2 B*
                    5 EΔ                      6 FΔ

```

The sub-columns of Z, consisting of the corresponding elements of its sub-arrays, determine $\supset[1]Z$.

```

p>[2]Z←2 4 3  (>[2]Z)[1;;]←1 A*      (>[2]Z)[2;;]←5 EΔ
                    2 B*                      6 FΔ
                    3 C*                      7 GΔ
                    4 D*                      8J9 HΔ

```

This time the rows of $\supset Z$ consist of the corresponding elements of the row sub-arrays of Z.

The monadic function ENLIST (APL character ϵ) will convert any array into a simple scalar array (a vector of scalars) after removing all the nesting. The effect of this function is different from that of the RAVEL function " , " which stretches out any array into a 1-dimensional array with all nested objects left intact.

```

εZ←1 2 3 4 A B C D * * * * 5 6 7 8J9 E F G H Δ Δ Δ Δ
pεZ←24          p,Z←6          p'',Z←4 4 4 4 4

```


FIRST is a monadic function, invoked with " ↑ " (same as TAKE) that selects the first element of an array, in row major order.

(↑Z)←'XXXX' will replace (1 2 3 4) in the first row, first column of Z with the character array XXXX.

The dyadic function MATCH (" = ", same as DEPTH) will yield a 1 if the left argument L matches the right argument R exactly, and a 0 otherwise.

'ABCD'='A' 'B' 'C' 'D' ↔ 1

'AB' 'CD'='ABCD' ↔ 0

'AB' 'CD'='AB CD' ↔ 0 since L is nested and R is not.

' '=10 ↔ 0 since L is a character and R is numeric.

The FIND function € is a dyadic function that yields a boolean array of the same shape as that of the right argument R with a 1 in each position where the pattern defined in the left argument L begins to occur in R.

'*****'€Z ↔ 0 0 1
 0 0 0

'XY'€'XYXYXYXYXY' ↔ 1 0 1 0 1 0 1 0 1 0

This function is different from MEMBER, APL character €, which produces a boolean array of the same shape as L with a 1 for every component of L that occurs in R.

APL2 OPERATORS

The EACH operator " " " was discussed briefly earlier. This operator can be combined with several primitive functions as well as user defined functions to create new user defined functions such as " +/" " which will sum each of the sub-arrays in a nested numerical array, separately. The two operators " / " and " \ " have new uses in APL2 as demonstrated in the examples below. Consider

X1 ↔ 1 2 3 4 X2 ↔ 1 2 3 4 Y1 ↔ ABCD

The next four examples illustrate how to EXPAND an array by replicating each element the same or different number of times.

2/X1 ↔ 1 1 2 2 3 3 4 4 replicates the elements of X1.

2 3/X2 ↔ 1 2 1 2 3 4 3 4 3 4

0 2/X2 ↔ 3 4 3 4

1 2 3 4/Y1 ↔ ABBCCDDDD

The next group of examples show some of the different ways to REDUCE a given array.

2+/X1 ↔ 3 5 7 the sums of successive pairs.
 2-/X1 ↔ -1 -1 -1 the differences of successive pairs.
 -2-/X1 ↔ -(2-/X1) ↔ 1 1 1
 3x/X1 ↔ 6 24 the products of successive triples.

Examples of the use of the primitive function " , " with the operator " / ".

,/X1 ↔ ,/X2 ↔ cX1 all scalar objects.
 4,/X1 ↔ 2,/X2 ↔ (,cX1) an array of shape 1.
 1,/X1 ↔ X1 1,/X2 ↔ X2
 2,/X1 ↔ 1 2 2 3 3 4 a mixed array of successive pairs.
 4,/X1 ↔ 2,/X2 and ρ2,/X2 = ρ4,/X1 = 1

Examples of reduction of higher dimensional arrays:

,/Z ↔ 1 2 3 4 ABCD**** 5 6 7 8J9 EFGHΔΔΔΔ
 ρ,/Z ↔ 2 and ρ",/Z ↔ 12 12
 ,[1]Z ↔ 1 2 3 4 5 6 7 8J9 ABCDEFGH ****ΔΔΔΔ
 ρ,[1]Z ↔ 3 and ρ",/[1]Z ↔ 8 8 8
 ,[2]Z ↔ ,/Z since the outermost axis reduction is the default.

Examples of the different ways of using the SCAN operator:

```
,\X1 ↔ 1 1 2 1 2 3 1 2 3 4  
ρ,\X1 ↔ 4 and ρ'',\X1 ↔ 1 2 3 4  
,\X2 ↔ 1 2 1 2 3 4  
ρ,\X2 ↔ 2 and ρ'',\X2 ↔ 2 4
```

In addition to the primitive operators, one can create user defined operators with unspecified operands, which result in generic user defined functions. In a particular application, specific primitive functions are used as operands in the invocation command. The user defined generic operator AND (selected from the EXAMPLES workspace in Public Library 1) is one which will perform two unspecified functions in parallel, e.g., the sum and product of two numbers. The operator can be created using any of the editors available in APL2 (discussed later on in this report). This example also illustrates the creation of an operator (or a function) that can be used either monadically or diadically; a new APL2 system variable DNC (discussed later on in this tutorial) is very useful for this purpose.

```
[0] Z←L (LF AND RF) R  
[1] →(0=DNC 'L')/V1  
[2] ⍝ APPLY TWO FUNCTIONS TO PRODUCE A PAIR OF RESULTS  
[3] ⍝ EXAMPLE: 4 +AND× 5  
[4] ⍝ DYADIC USAGE
```

[5] $Z \leftarrow (\leftarrow L \text{ LF } R), \leftarrow L \text{ RF } R$

[6] $\rightarrow 0$

[7] A MONADIC USE

[8] $V1: Z \leftarrow (\leftarrow L \text{ LF } R), \leftarrow R \text{ RF } R$

The operands LF and RF in the above definition can be any two APL2 functions; specific choices for these functions are made at the time of application of the operator.

EXAMPLES OF THE USE OF THE OPERATOR 'AND'

$4 + \text{AND} \times 5 \leftrightarrow 9 \ 20$ (dyadic use of the operator)

$-\text{AND} \div 5 \leftrightarrow -5 \ 0.2$ (monadic use of the operator)

In the first example, the primitive functions " + " and " x " have been selected for LF and RF respectively. In the second example, the negation function " - " and the reciprocal function " \div " are the left and right operands of the operator AND. Also in the second example, the operator is used monadically i.e., only a right argument is used with the operator. The first example illustrates the dyadic use of the same operator.

SYSTEM VARIABLES AND SYSTEM FUNCTIONS

A number of new system variables and functions are included in APL2. A few of these that are important for standard applications are discussed below.

`DDL R` (Delay function)

R is a numeric scalar. This function will cause a delay of R seconds between the execution of two successive APL expressions or commands.

`L DEB R` (Execute Alternate)

L and R are character arrays containing valid APL expressions e.g., 'A \circ .xB' or 'DESCRIBE' etc. The effect of this function is to execute the APL expression R. If the execution is completed without interruption the result is displayed. Otherwise, the alternative expression in L will be executed. For example, '13' DEB '14' will result in the display of the array (1 2 3 4) and '13' DEB '14.5' will result in the evaluation of 13 since 14.5 is undefined.

`OL and OR` (Left and Right Arguments)

Whenever a primitive dyadic function is suspended

either in the immediate execution mode or within a defined function, the left argument of the dyadic function is temporarily stored in $\square L$ and the right argument is stored in $\square R$; for a monadic function only $\square R$ will be created. One or both of these system variables can now be assigned new values that will make the function executable. In the immediate execution mode (calculator mode) the command: $\rightarrow 0$ will then complete the execution of the corrected expression. The analogous command for a suspended function is: $\rightarrow N$, where N is the line number where the function is pendant, to complete the execution of the function. The APL expression $2\ 3+4\ 5\ 6$ will result in a LENGTH ERROR since the two argument arrays are of different shapes. $\square L$ will now contain the left argument array (2 3) and $\square R$ holds the right argument array (4 5 6). The expression can now be made executable either by assigning to $\square L$ a 3-array or by assigning to $\square R$ a 2-array or by assigning to $\square L$ and $\square R$ any two arrays of the same length or shape. Thus, for example, $\square L \leftarrow 1\ 2\ 3$ followed by the command: $\rightarrow 0$ will display the array (5 7 9) the result of adding the arrays (1 2 3) and (4 5 6).

DNC R

(Name Class)

R is a character array (scalar, vector or matrix) containing the names of APL objects. This function will check each of the names in R to see if it is already in use in the workspace; a 1, 2, 3 or a 4 will be displayed if the name represents a label, variable, function or an operator, respectively. If a name is not in use but conforms to APL naming conventions a 0 is returned and a -1 implies that the name is invalid. DNC was used in the definition of the operator AND discussed earlier.

L DNL R

(Name List)

The left argument L is optional and may be omitted. The array R can be any subset of the array (1 2 3 4). The names of APL objects specified in R (labels if R=1, variables if R=2, functions if R=3 and operators if R=4) are displayed. If L is included in the command, only those names that begin with the alphabets included in L, will be listed. 'AB' DNL 2 3 will display the names of all variables and functions that begin with the letters A or B.

NEW SYSTEM COMMANDS

OUT FILENAME NAMELIST

This function creates on the A-disk a "transfer file", with the specified file name and file type APLTF, that contains the APL objects specified in the namelist, which is optional. If the namelist is omitted all user defined objects and certain system variables are transferred. This command can be very useful in situations where the workspace is nearly full and more room is needed. Some of the objects that are not immediately needed can then be transferred and the objects erased to generate more room.

IN FILENAME NAMELIST

This command will result in the creation, in the active workspace, of the objects in the namelist from the transfer file created with the OUT command. The namelist can be a subset of the one used with the OUT command and if it is omitted the entire transfer file will be copied into the workspace.

NMS

The names of all user defined variables, functions

and operators will be listed. Each of the names ends with a period followed by a 2, 3 or a 4 to indicate if it is a variable, function or operator name. Optionally, the beginning and ending alphabet sequence for the names to be listed, may be included in the command; JNMS BE EXA will list only those names that begin with the sequence BE through EXA.

Sometimes it is convenient to assign a group name to those objects that share a common trait. All that it takes to create a group is to define a character matrix with the names of the objects to be included in the group as rows e.g.,

GROUP1←>'VAR1' 'OP2' 'FN5' which creates a group with three objects. The name of the group itself may be included as one of the names in the matrix; this would be useful when erasing the group or when copying the group into another workspace. To apply an APL command to an entire group, the group name must be specified within parentheses. If the command applies only to a subset, the group name without parentheses followed by the list of objects, must be specified. For example, the command: JERASE (GROUP1) will erase the three objects VAR1, OP2 and FN5. However, JERASE GROUP1 OP2 will only erase the one object OP2 and GROUP1 will now have

just two objects. Also, the)NMS command will list the group name as one of the variable names.

)OPS

Lists the names of all user defined operators. Here again, starting and ending alphabet sequences may be specified.

)QUOTA

A six element array whose components are the total amount of virtual memory, virtual memory still unused, the workspace size, workspace still available, size of the memory reserved for shared storage and the maximum number of shared variables that may be defined, is displayed.

)PBS ON

PBS stands for printable backspace. This command is necessary to generate "overstruck" characters such as **■**, **▲**, **▼**, **=** on terminals that do not have separate individual keys for such characters. After the command has been issued, an overstruck character is generated by typing in sequence the two component characters (e.g., **□** and **+**) with the underbar character " **_** " (shift F) between them.

Thus, the sequence " Δ _ " is equivalent to the overstruck character " Δ " , " = _ _ " is the APL character for the DEPTH function and " $\square \div$ " is the same as the character \boxplus .

)RESET N

Clears the first N lines from the State Indicator. If N is omitted the entire State Indicator is cleared.

)HOST CMS-COMMAND

The specified CMS command will be executed and control is returned to the APL2 environment. If a CMS command is not specified, the name of the host operating system (CMS at NPS) will be displayed. The command)HOST SUBSET will temporarily create a CMS sub-environment within APL2; return to APL2 is achieved with the command: EXIT.

)MSGN USERID MESSAGE

The message is sent to the user with the specified userid.

APL2 EDITORS

Three different editors viz., a line editor , a full screen editor and the editor associated with the computer operating system (XEDIT at NPS) are available for use in APL2. The respective commands for selecting these editors are

)EDITOR 1	for the Line Editor.
)EDITOR 2	for the Full Screen Editor.
)EDITOR XEDIT	for XEDIT.

There will not be any system response when one of these commands is issued unless an error is detected. The line editor is essentially the same as the one in VSAPL and will not be discussed in this report. The system editor XEDIT is one of the choices for a full screen editor and it can be used to create or modify APL functions, character variables and character matrices. Only currently existing APL character variables and matrices can be edited with XEDIT. In order to create a new such object, the first step is to define a dummy variable or matrix with a specified name in the workspace (for example: VARNAME←'' will create an empty character variable); it can then be modified using the editor. The editing process is initiated with the command: VVARNAME or VFNNNAME or VZ←L FNNNAME R etc. If APL characters are included in the object being edited, issue the command: SET APL ON in XEDIT to insure that they are properly displayed. All of the XEDIT commands will now be available to the user. To add new lines at the end of an existing object the INPUT command (I) may be used. However, it is very

important that the BOTTOM command (BOT) is issued prior to the INPUT command. Otherwise all new lines will be written above the "HEADER LINE" and it will not be possible to save the object or even to abandon the editing process; turning the power off may be the only way out.

The APL2 full screen editor is quite different from the VSAPL editor. The new editor does not provide a command line at the bottom of the screen to enter editing commands. All commands are entered by typing certain APL characters between the square brackets of any displayed line of the object being edited. When the Enter key is pressed, the command is executed and the original line will be restored. The following are some of the full screen editor commands; unless stated otherwise these commands may be entered on any displayed line.

[→]	Abandon the edit session.
[▽]	Save the object; remain in editor (PF6).
▽	This command is entered at the end of any displayed line or on a new line to save the object and exit from the editor (PF3).
[Δn1-n2]	Delete lines n1 through n2 inclusive.
[Δn1,n2,...]	Delete the specified lines n1, n2 etc.
[Δ-n]	Delete lines 1 through n.
[Δn-]	Delete lines n through the last line.
[Δ]	Delete all lines except the header.
[!]]	Renumber the lines (PF2).

[v name n1-n2] Save (PUT) lines n1 through n2 under the specified name for later reuse. The name is optional unless multiple PUTs are planned. A different name must be used with each PUT command; otherwise, the file will contain only the objects from the most recent PUT command.

[^ name n1-n2] Retrieve (GET) specified lines saved with the PUT command.

[] This command (blank line number) on any line implies that the text on the line is a continuation of the preceding line.

[r] Scroll forward one screen starting with the cursor line (PF9).

[↑] Scroll back to the preceding screen (PF7).

[↓] Scroll forward to the next screen (PF8).

[?] Display the PF key settings.

[*]expression This command will result in a temporary exit from the editor to execute the APL expression; the expression will not be a part of the object being edited.

New lines may be added at the end of an existing object in one of three ways: (1) type on a blank line after the last displayed line or (2) type over the header line starting from the left line number bracket ([) or (3) type over any existing line after replacing its line number with that of the new line. In the latter

two cases, when the Enter key is pressed the new line will be created at the end and the original lines will be restored intact.

Any existing line may be modified by typing over the existing text or by typing the desired line number followed by the modified text on a blank line at the end. An existing line may be duplicated as a new line by replacing its line number with the desired new line number; on pressing the Enter key the old line will be restored and the new line is created.

A new line of text may be created between lines n and $n+1$ by typing the new text on line $n+1$ starting from the left bracket ([) of the line number. Pressing the Enter key will create the new line with a fractional line number e.g., [3.1] and the $(n+1)$ st line will be restored. If desired, pressing PF2 key will renumber the lines immediately; otherwise, the renumbering will be done automatically at the end of the edit session.

More than one object can be edited simultaneously with the APL2 full screen editor. While editing OBJECT1 say, the command: VOBJECT2 on any displayed line of OBJECT1 (no text to the left or right of the command) will display the second object for editing starting from the line at which the command was issued. The lines of the first object that have been overwritten may still be displayed using scrolling. If desired a third object may be brought in by following the same procedure, etc. When done with editing an object, the object may be closed with the command: ▽ at

the end of any displayed line of the object and the immediately preceding object will become the current object for editing.

AUXILIARY PROCESSORS

Several auxiliary processors (APs) that allow the interfacing of non-APL programs with APL programs are available to the APL2 users. These APs have many uses such as reading and writing to CMS files, accessing peripherals and running non-APL programs such as Fortran programs from an APL environment. Most of these processors are automatically attached when the APL2 command is issued. One of these APs, AP 121 (APL2 Data File Processor) will be discussed here in detail; two others viz., AP 100 (Host System Command Processor) and AP 101 (Alternate Input Processor) will be briefly mentioned. The Host System Command Processor (AP 100) allows the use of CP or CMS commands; the APL command:)HOST CMS-COMMAND will execute the CMS-COMMAND and returns the user to the APL environment. The command:)HOST SUBSET creates CMS as a sub-environment of APL. Entering the command: RETURN will reactivate the APL environment. The Alternate Input Processor (AP 101) can be used to run non-APL programs such as Fortran programs and use its output as input to an APL function. This processor can also be used to run an APL program from the CMS environment. Details on the use of these processors are in reference [2].

The APL2 Data File Processor (AP 121) allows the creation, on the A-disk, of an APL file (file type VSAPLFL) containing temporarily unneeded APL objects such as data sets, variables and functions; these objects can be retrieved from the file when needed. This capability is very useful in situations where several different large data sets need to be processed and the active workspace does not have enough room. Also, because of the structure of an APL file it requires much less disk space to save an APL file than an APL workspace that contains the same objects. Two types of files, sequential or direct access, may be created using this processor. Objects written to these files, called records, are assigned consecutive numbers starting from 1 for purposes of identification. Records from a direct file can be retrieved in any order and sequential file records can only be read on a first in first out basis; the 10th record in a sequential file can only be accessed after the preceding 9 records are read. There are no size limitations on records written to a sequential file but when creating a direct file the size of the largest record (maximum allowed is 4054 bytes) that will be written, must be specified. The actual process of creating files and retrieving records is achieved using "shared variables" and system functions/variables such as `DSVO`, and `DSVC`. Two shared variables, a "control variable" whose name must start with CTL and a "data variable" with a name that must start with DAT are needed to interact with the AP 121 processor, and a prescribed dialogue must be carried out to successfully create a file and to add or retrieve records. The following example illustrates the sequence of commands needed to

create a direct file called TESTFILE and to add certain fictitious records to it.

121 DSVO" 'CTL1' 'DAT1' Offer to share the two variables.

2 2 This is the system response to indicate acceptance of the proposed variables. A return of a 0 indicates that the sharing was unsuccessful; a 1 indicates that the offer is pending.

CTL1←'C TESTFILE D' Request the creation of a direct file named TESTFILE; CTL1←'C TESTFILE S' will create a sequential file.

CTL1 Check for successful creation of the file.

0 OK. Any number other than 0 indicates failure; see reference [2] for the meanings of other responses.

CTL1←'SWC TESTFILE 100' Open the file for sequential writing of records. Records are written sequentially in both types of files. The number 100 indicates the size of of the largest record.

CTL1 Check for successful execution.

0 OK.

CTL1←VAR1 Write the APL variable VAR1 as the first record.

CTL1 Check

0 OK

CTL1←FN1 Write the function FN1

	as the second record.
CTL1	Check
0	OK
CTL1←'JIM SMITH'	Write the third record, JIM SMITH.
CTL1	Check
0	OK
CTL1←Z	Write the fourth record, the nested variable Z.
CTL1	Check.
0	OK
CTL1←''	Close the file.
CTL1	Check.
0	OK
CTL1←'DR TESTFILE'	Open the file for direct reading.
CTL1	Check.
0	OK
CTL1←0,3	Read the third record and store it in DAT1, the shared variable created earlier.
CTL1	Check. This check is optional but is a good idea.
0	OK
NAME←DAT1	Creates the character variable NAME which contains JIM SMITH.
CTL1←0,1	Read the first record into DAT1.
CTL1	Check.
0	OK

ZZ←DAT1	ZZ is assigned the contents of Var1.
CTL1←''	Done with reading from the file.
CTL1	Check.
0	OK
The commands for reading from sequential files are as follows.	
CTL1←'SR TESTFILE'	Open the file for sequential reading.
CTL1	Check.
0	OK
Var1←CTL1	Var1 is recreated in the workspace.
FN1←CTL1	The function FN1 is created in the workspace.
ZZ2←CTL1	The character variable ZZ2 containing JIM SMITH is created.
CTL1←''	Finished reading.
CTL1	Check.
0	OK

PUBLIC LIBRARIES

Two public libraries supplied with APL2 (LIB 1 and LIB 2) contain the following workspaces:

LIB 1	-	DISPLAY, EXAMPLES, MATHFNS, MEDIT,
		UTILITY, WSINFO
LIB 2	-	APLDATA, CMS, CMSIVP, FSC126, FSM,

GDMX, GRAPHPAK, PRINTWS, SQL, TRANSFER,
VAPLFILE, VSAMDATA

Each of these workspaces contains ABSTRACT, DESCRIBE, and HOW variables that provide information about the workspace. Also, the workspace WSINFO in LIB 1 has brief descriptions on all the workspaces in both libraries. The more important workspaces are EXAMPLES, MATHFNS, UTILITY, APLDATA, CMS and PRINTWS. The EXAMPLES workspace contains examples on creating user defined operators and functions. MATHFNS has functions to perform certain mathematical computations such as finding eigenvalues, finding roots of polynomials etc. UTILITY is similar to EXAMPLES and contains various functions to perform frequently used operations. APLDATA is made of functions to create and read from APL files, that automatically carry out the necessary dialogue described in the previous section. The CMS workspace includes functions for interacting with the CMS operating system and to run non-APL programs. PRINTWS may be used to print entire APL workspaces or specified APL functions and variables on the computing center's laser printer.

FINAL COMMENTS

A new release of APL2 (release 3), currently under test, has several new features such as the ability to create and edit numeric as well as nested arrays, no size restrictions on direct files, and

new nested array handling capabilities. This version can be invoked with the command APL2T instead of APL2; this release requires a minimum of 2 megabytes of virtual memory. It is expected that release 3 will replace release 2, after the new mainframe computer is acquired and installed.

REFERENCES

- [1] "APL2 Programming: Language Reference",
IBM Order No.SH20-9227-1, Program No.5668-899,
Release 2.
- [2] "APL2 Programming: System Services Reference",
IBM Order No.SH20-9218-1, Program No.5668-899,
Release 2.
- [3] "APL2 At a Glance" by J.A. Brown, S. Pakin and
R.P. Polivka, Prentice Hall, New Jersey, 1988.

DISTRIBUTION LIST

DIRECTOR (2)
DEFENSE TECH. INFORMATION
CENTER, CAMERON STATION
ALEXANDRIA, VA 22314

DIRECTOR OF RESEARCH ADM.
CODE 012
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93943

LIBRARY (2)
CODE 0142
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93943

DEPT. OF MATHEMATICS
CODE 53
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93943

PROF. TOKE JAYACHANDRAN (25)
CODE 53JY
DEPARTMENT OF MATHEMATICS
MONTEREY, CA 93943

CENTER FOR NAVAL ANALYSES
4401 Ford Ave.
Alexandria, VA 22302-0268